1 Le type `dict`

1.1 Définition des dictionnaires

Les données de type `dict`, également présentes dans d'autres langages sous le nom de « mémoires associatives » ou de « tableaux associatifs » sont des ensembles non ordonnés d'objets auxquels on accède à l'aide d'une *clé*.

Définition 1.1 Les *dictionnaires* sont des ensembles non ordonnés de paires `clé:valeur`.

Remarque 1.2 En particulier, les dictionnaires ne sont pas, à la différence des séquences, indexés par des nombres. Les clés peuvent être de n'importe quel type, à condition d'être immuables (ne peuvent pas être modifiées). On peut donc choisir comme clé des nombres et des chaînes de caractères, mais jamais des listes.

Exemple 1.3 `{'pomme':30, 'poire':22, 'banane':12}` est un dictionnaire dont les clés sont des chaînes de caractères et les valeurs des nombres.

Définition 1.4 (Longueur d'une dictionnaire) On appelle *longueur d'un dictionnaire* le nombre de paires `clé:valeur` qui le compose. Cette longueur peut-être obtenue à l'aide de la fonction `len`.

Exemple 1.5 L'instruction `len({'pomme':30, 'poire':22, 'banane':12})` renvoie `3`.

Définition 1.6 Il existe plusieurs manières de définir un dictionnaire :

- **Dictionnaire vide :** il est défini par l'instruction `{}` ou par l'instruction `dict()`
- **Dictionnaire par extension :** il est défini en indiquant directement entre accolades les éléments du dictionnaire.
  Exemple : le dictionnaire `{'pomme':30, 'poire':22, 'banane':12}` est un dictionnaire défini par extension.
- **Dictionnaire par compréhension :** il est défini en indiquant entre accolades la manière dont le dictionnaire est construit. On utilise pour cela l'une des syntaxes suivantes :
  - lorsque les clés et valeurs sont construites à partir d'une même structure itérable (types `list`, `tuple` ou `str`) :

    `{clé(elem):valeur(elem) for elem in itérable if condition(elem)}`

    où `clé(elem)`, `valeur(elem)` et `condition(elem)` sont des fonctions de paramètre `elem`.
    Exemple : les deux dictionnaires suivants sont des dictionnaires définis par compréhension :
    - `{x:x**2 for x in range(1,4)}` correspond au dictionnaire `{1:1, 2:4, 3:9}`
    - `{x:x**3 for x in range(10) if x%5 == 0}` correspond au dictionnaire `{0:0, 5:125}`
  - lorsque les clés et valeurs sont définies par deux structures itérables différentes :

    `{clé:valeur for clé, valeur in zip(iterable_clés,itérable_valeurs)}`

    On peut également ajouter des conditions après le `for`.
    Exemple : le dictionnaire `{cle:val for cle,val in zip(['pomme','poire'],(30,22))}` correspond au dictionnaire `{'poire':22, 'pomme':30}`.
- **Dictionnaire par transtypage :** il est défini en appliquant la fonction `dict` sur une liste de paires `(clé,valeur)` stockées sous la forme de tuples.
  Exemple : l'instruction `dict([('pomme',30), ('poire',22), ('banane',12)])` renvoie le dictionnaire `{'pomme':30, 'poire':22, 'banane':12}`

1.2 Manipuler les données d'un dictionnaire

⋆ Récupérer des données dans un dictionnaire `D` :

- `D[cle]` : renvoie la valeur de `D` dont la clé est `cle` quand elle existe, sinon une erreur `KeyError` se produit
- `D.get(cle,V)` : renvoie la valeur de `D` dont la clé est `cle` et la valeur `V` si la clé n'existe pas
- `D.keys()` : renvoie les clés présentes dans `D`
- `D.values()` : renvoie les valeurs présentes dans `D`
- `D.items()` : renvoie les paires `clé:valeur`, chaque paire étant un tuple

⋆ Modifier des données d'un dictionnaire `D` :

- `D[cle] = x` : si la clé `cle` existe déjà dans `D`, l'ancienne valeur est supprimée et remplacée par `x` ; si elle n'existe pas dans `D`, elle est créée et prend la valeur `x`
- `del(D[cle])` : supprime la paire (`cle`,valeur) de `D` ; une erreur `KeyError` se produit si la clé n'existe pas
- `x = D.pop(cle,V)` : supprime la paire (`cle`,valeur) de `D` et affecte la valeur à `x` ; si la clé n'existe pas, la valeur `V` est alors affectée à `x`
- `x = D.popitem()` : supprime arbitrairement une paire (clé,valeur) de `D` et l'affecte à `x` sous la forme d'un tuple ;
- `D.clear()` : supprime tous les éléments de `D` et renvoie un dictionnaire vide

⋆ Ajouter à un dictionnaire `D` des données issues d'un autre dictionnaire `E` :

- `D.update(E)` : ajoute à `D` les paires `clé:valeur` du dictionnaire `E` ; si une clé de `E` existe déjà dans `D`, alors la valeur associée est mise à jour avec celle de `E`

⋆ Copier un dictionnaire `D` :

- `D.copy()` : renvoie une copie indépendante du dictionnaire `D`

1.3 Tests

⋆ On peut tester si deux dictionnaires sont égaux ou différens grâce aux opérateurs de comparaison == et `!=`
Exemples :

- L'instruction `'pomme':12, 'poire':30} == {'poire':30, 'pomme':12}` renvoie `True` (les deux dictionnaires sont égaux).
- L'instruction `{'pomm'':12, 'poire':30} != {'poire':30, 'pomme':12}` renvoie donc `False`.
- L'instruction `{'pomme':12, 'poire':30} == {'poire':35, 'pomme':12}` renvoie `False` (les deux dictionnaires ne sont pas égaux).
- L'instruction `{'pomme':12, 'poire':30} == {'poire':35, 'pomme':12}` renvoie donc `True`.

⋆ On peut également tester si une clé est présente dans un dictionnaire ou non grâce aux opérateurs `in` et `not in`
Exemples :

- L'instruction `'pomme' in {'pomme':12, 'poire':30}` renvoie `True` (`'pomme'` est une clé présente).
- L'instruction `'pomme' not in {'pomme':12, 'poire':30}` renvoie donc `False`.
- L'instruction `'fraise' in {'pomme':12, 'poire':30}` renvoie `False` (`'fraise'` n'est pas une clé existante).
- L'instruction `'fraise' in {'pomme':12, 'poire':30}` renvoie donc `True`.

2 Parcours de dictionnaires

Un dictionnaire est une structure qui peut être parcourue afin d'obtenir des informations sur chacun de ses éléments.

Il existe trois types de parcours : le **parcours par clé**, le **parcours par valeur** et le **parcours par élément**.

Définition 2.1 On donne ci-dessous la syntaxe `Python` pour chacun des trois parcours pour un dictionnaire `D` :

- Parcours par clé :

```
for cle in D:
    bloc d'instructions
```

ou

```
for cle in D.keys():
    bloc d'instructions
```

- Parcours par valeur :

```
for val in D.values():
    bloc d'instructions
```

- Parcours par élément :

```
for elem in D.items():
    bloc d'instructions
```

ou

```
for cle, val in D.items():
    bloc d'instructions
```

Dans la première syntaxe, `elem` est un tuple de la forme `(clé, valeur)` ; dans la seconde syntaxe, `cle` parcourt les clés et `val` les valeurs associées.

Exemple 2.2 On considère le dictionnaire `D = {'pomme':30, 'poire':22, 'banane':12}`.

Parcours par clé : le code

```
for cle in D:
    print(cle)
```

affiche

```
'pomme'
'poire'
'banane'
```

Parcours par valeur : le code

```
for val in D.values():
    print(val)
```

affiche

```
30
22
12
```

Parcours par élément : le code

```
for elem in D.items():
    print(elem)
```

affiche

```
('pomme', 30)
('poire', 22)
('banane', 12)
```

3 Exercices

Exercice 3.1 Compléter le Notebook *NSI Première Partie 1 Chapitre 9 Dictionnaires*.

Exercice 3.2 (QCM)

1. On considère le dictionnaire `D = {'A':2, 'B':1, 'C':5}`. Que renvoie l'instruction `len(D)` ?
   (a) une erreur (b) une somme (c) `3` (d) `6`

2. On considère le dictionnaire `D = {'A':2, 'B':1, 'C':5}`. Que renvoie l'instruction `'a' not in D` ?
   (a) rien (b) une erreur (c) `True` (d) `False`

3. On considère le dictionnaire `D = {'A':2, 'B':1, 'C':5}`. Que renvoie l'instruction `'2' in D.values()` ?
   (a) rien (b) une erreur (c) `True` (d) `False`

4. On exécute l'instruction `D['A'] = 3` sur un dictionnaire `D`. Parmi les affirmations suivantes, lesquelles sont vraies ?
   (a) si `'A'` est une clé de `D`, alors sa valeur est mise à jour avec la valeur `3`
   (b) si `'A'` n'est pas une clé de `D`, alors elle est créée avec la valeur associée égale à `3`
   (c) si `3` est une clé de `D`, alors sa valeur est mise à jour avec la valeur `'A'`
   (d) si `3` n'est pas une clé de `D`, alors elle est créée avec la valeur associée égale à `'A'`

5. On exécute l'instruction `D[['A']] = 3` sur un dictionnaire `D`. Parmi les affirmations suivantes, lesquelles sont vraies ?
   (a) si `['A']` est une clé de `D`, alors sa valeur est mise à jour avec la valeur `3`
   (b) si `3` est une clé de `D`, alors sa valeur est mise à jour avec la valeur `['A']`
   (c) cette instruction ne fait rien
   (d) cette instruction déclenche une erreur

6. On considère la fonction suivante :

```
def comptage(D, val):
    ''' D est de type dict '''
    cpt = 0
    for cle in D:
        if D[cle] == val:
            cpt += 1
    return cpt
```

   Parmi les instructions suivantes, lesquelles sont vraies ?
   (a) `comptage({'A':2, 'B':1, 'C':2, 'D':2}, 2)` renvoie `3`
   (b) `comptage({'A':2, 'B':1, 'C':2, 'D':2}, 2)` renvoie `'A'`
   (c) cette fonction renvoie une erreur si les valeurs de `D` ne sont pas numériques
   (d) cette fonction renvoie le nombre de clés de `D` dont les valeurs associées sont égales à `val`

7. On considère la fonction suivante :

```
def seuil(D):
    ''' D est de type dict '''
    for cle in D:
        if D[cle] > 127:
            D[cle] = 255
    return D
```

   Parmi les instructions suivantes, lesquelles sont vraies ?
   (a) `seuil({'A':200, 'B':127, 'C':30, 'D':129})` renvoie `{'A':255, 'B':255, 'C':30, 'D':255}`
   (b) `seuil({'A':200, 'B':127, 'C':30, 'D':129})` renvoie `{'A':255, 'B':127, 'C':30, 'D':255}`
   (c) `seuil({'A':200, 'B':127, 'C':30, 'D':129})` renvoie `{'A':255, 'B':0, 'C':0, 'D':255}`
   (d) cette fonction renvoie une erreur car un dictionnaire n'est pas modifiable